\( \text{asinh } a \) \( \rightarrow \text{asinh} \), \( \text{acosh} \), or \( \text{atanh} \), respectively.

\( \text{cis } a \) \( \rightarrow \cos a + i \sin a \).

\( \text{conjugate } a \) \( \rightarrow \) Return complex conjugate of \( a \).

\( \text{max } \text{num}^+ \) \( \rightarrow \) Greatest or least, respectively, of \( \text{num}^+ \).

\( \text{mod } \), \( \text{rem } \) \( \rightarrow \) Same as \( \text{floor} \) or \( \text{truncate} \), respectively, but return remainder only.

\( \text{random } \text{limit} \) \( \rightarrow \) Return non-negative random number less than \( \text{limit} \), and of the same type.

\( \text{make-random-state} \) \( \rightarrow \) Copy of \( \text{random-state} \) object \( \text{state} \) or of the current random state; or a randomly initialized fresh random state.

\( \text{float-sign } \text{num-a} \) \( \rightarrow \text{num-a} \) with \( \text{num-b} \)’s sign.

\( \text{signum } n \) \( \rightarrow \) Number of magnitude 1 representing sign or phase of \( n \).

\( \text{numerator } \text{rational} \) \( \rightarrow \) Numerator or denominator, respectively, of \( \text{rational} \)’s canonical form.

\( \text{realpart } \text{number} \) \( \rightarrow \) Real part or imaginary part, respectively, of \( \text{number} \).

\( \text{imagpart } \text{number} \) \( \rightarrow \) Make a complex number.

\( \text{real} \) \( \rightarrow \) Angle of \( \text{num} \)’s polar representation.

\( \text{abs } n \) \( \rightarrow \) Return \( |n| \).

\( \text{rationalize } \text{real} \) \( \rightarrow \) Convert \( \text{real} \) to \( \text{rational} \). Assume complete/limited accuracy for \( \text{real} \).

\( \text{float } \text{real} \) \( \rightarrow \) Convert \( \text{real} \) into \( \text{float} \) with type of \( \text{prototype} \).

1.3 Logic Functions

Negative integers are used in two’s complement representation.

\( \text{boole } \text{operation} \) \( \text{int-a} \) \( \text{int-b} \) \( \rightarrow \) Return value of bitwise logical \( \text{operation} \). \( \text{operations} \) are

\( \text{.boole-1} \) \( \rightarrow \) \( \text{int-a} \).

\( \text{.boole-2} \) \( \rightarrow \) \( \text{int-b} \).

\( \text{.boole-c1} \) \( \rightarrow \) \( \neg \text{int-a} \).

\( \text{.boole-c2} \) \( \rightarrow \) \( \neg \text{int-b} \).

\( \text{.boole-set} \) \( \rightarrow \) All bits set.

\( \text{.boole-clr} \) \( \rightarrow \) All bits zero.
Contents

1 Numbers  3  9.5 Control Flow . . . . 19
1.1 Predicates . . . . 3  9.6 Iteration . . . . 21
1.2 Numeric Functions . 3  9.7 Loop Facility . . . . 21
1.3 Logic Functions . 4  10 CLOS 24
1.4 Integer Functions . 5  10.1 Classes . . . . 24
1.5 Implementation- 6  10.2 Generic Functions 25
dependent . . . .
2 Characters 6
3 Strings 7  11 Conditions and Errors 27
4 Conses 8  12 Types and Classes 30
4.1 Predicates . . . . 8  4.1.1 Predicates . . . . 42
4.2 Lists . . . . 8  4.1.2 Predicates . . . . 42
4.3 Association Lists 9  4.1.3 Predicates . . . . 42
4.4 Trees . . . . 10  4.1.4 Predicates . . . . 42
4.5 Sets . . . . 10  4.1.5 Predicates . . . . 42
5 Arrays 10
5.1 Predicates . . . . 10  5.2 Array Functions . . . . 42
5.2 Array Functions . 10  5.3 Vector Functions . . . . 42
5.3 Vector Functions . 11
6 Sequences 12  14 Packages and Symbols 42
6.1 Seq. Predicates . . 12  6.1.1 Predicates . . . . 42
6.2 Seq. Functions . . 12  6.1.2 Predicates . . . . 42
6.3 Lists . . . . 12  6.1.3 Predicates . . . . 42
6.4 Trees . . . . 13
7 Hash Tables 14
8 Structures 15  15 Compiler 44
9 Control Structure 15
9.1 Predicates . . . . 15
9.2 Variables . . . . 16
9.3 Functions . . . . 17
9.4 Macros . . . . 18
16 External Environment 47

Typographic Conventions

name; _name; _name; _name; _name; _name; _name
 ▶ Symbol defined in Common Lisp; esp. function, generic
function, macro, special operator, variable, constant.

them  ▶ Placeholder for actual code.

me  ▶ Literal text.

/foo   ▶ Either one foo or nothing; defaults to bar.
/foo*  ▶ Zero or more foos.
/foo+  ▶ One or more foos.
/foo  ▶ English plural denotes a list argument.

/foo|bar|baz| {foo|bar|baz};
 ▶ Anything from none to each of foo, bar, and baz.
/baz

/foo  ▶ Argument foo is not evaluated.
/bar  ▶ Argument bar is possibly modified.
/foo*  ▶ foo* is evaluated as in ; see page 20.

/foo; bar; baz  ▶ Primary, secondary, and nth return value.
/T; NIL  ▶ t, or truth in general; and nil or ()

1 Numbers

1.1 Predicates

\(\equiv\) number  ▶ Return \(\sum a\) or \(\Pi a\), respectively.
\(\neq\) number  ▶ Return \(-b\) or \(\Pi b\), respectively. Without any bs, return \(-a\) or \(\Pi a\), respectively.
\(\geq\) a  ▶ Return \(a \geq \sum b\) or \(\Pi b\), respectively. Without any bs, return \(-a\) or \(\Pi a\), respectively.
\(\leq\) a  ▶ Return \(a + 1\) or \(a - 1\), respectively.
\(\lcm a\)  ▶ Least common multiple of integers. \((\gcd a\) returns \(0\))
\(\gcd a\)  ▶ Least common multiple or greatest common denominator, respectively, of integers. \((\gcd a\) returns \(0\))
\(\pi\)  ▶ long-float approximation of \(\pi\), Ludolph's number.
\(\sin a\)  ▶ \(\sin a, \cos a, or tan a\), respectively. (a in radians.)
\(\tan a\)  ▶ \(\arcsin a\) or \(\arccos a\), respectively, in radians.
\(\arctan a\)  ▶ \(\arctan a\) or \(\arccos a\) in radians.
\(\sinh a\)  ▶ \(\sinh a, \cosh a\), or \(\tanh a\), respectively.
\(\tanh a\)  ▶ \(\sinh a, \cosh a\), or \(\tanh a\), respectively.
4 Conses

4.1 Predicates

(=cons foo bar)  Return T if foo is a cons of indicated type.

(=listp foo)  Return T if foo is NIL.

(=null foo)  Return T if foo is not a cons.

(=tailp foo list)  Return T if foo is a tail of list.

(=member foo list)  Return tail of list starting with its first element matching foo. Return NIL if there is no such element.

(=member-if list [key function])  Return tail of list starting with its first element satisfying test. Return NIL if there is no such element.

(=subset list-a list-b)  Return T if list-a is a subset of list-b.

4.2 Lists

(=car foo bar)  Return new cons {foo , bar}.

(=list foo*)  Return list of foos.

(=rest foo)  Return list of foos with last foo becomingcdr of last cons. Return foo if only one foo given.

(=make-list num [initial-element for])  New list with num elements set to for.

(=list-length list)  Length of list; NIL for circular list.

(=car list)  Car of list or NIL if list is NIL. setfable.

(=cdr list)  Cdr of list or NIL if list is NIL. setfable.

(=nthcdr n list)  Return tail of list after calling cdr n times.

(=first one second third fourth fifth sixth . . . ninth tenth list)  Return nth element of list if any, or NIL otherwise. setfable.

(=nth n list)  Zero-indexed nth element of list. setfable.

(=rXr list)  With X being one to four as and ds representing cars and cdrs, e.g. (=r cdr bar) is equivalent to (=car (=r cdr bar)). setfable.

(=last list [num])  Return list of last num conses of list.

1.4 Integer Functions

(=integer-length integer)  Number of bits necessary to represent integer.

(=ash integer count)  Return copy of integer arithmetically shifted left by count adding zeros at the right, or, for count < 0, shifted right discarding bits.

(=integer-length integer)  Return number of bits in integer.

(=make-list num [initial-element for])  New list with num elements set to for.

(=list-length list)  Length of list; NIL for circular list.

(=car list)  Car of list or NIL if list is NIL. setfable.

(=cdr list)  Cdr of list or NIL if list is NIL. setfable.

(=nthcdr n list)  Return tail of list after calling cdr n times.

(=first one second third fourth fifth sixth . . . ninth tenth list)  Return nth element of list if any, or NIL otherwise. setfable.

(=nth n list)  Zero-indexed nth element of list. setfable.

(=rXr list)  With X being one to four as and ds representing cars and cdrs, e.g. (=r cdr bar) is equivalent to (=car (=r cdr bar)). setfable.

(=last list [num])  Return list of last num conses of list.
1.5 Implementation-Dependent

- `short-float`, `single-float`, `double-float`, `long-float`
  - Smallest possible number making a difference when added or subtracted, respectively.
- `least-negative`, `least-negative-normalized`, `least-positive`, `least-positive-normalized`
  - Available numbers closest to $-\infty$ or $+\infty$, respectively.
- `most-negative`, `most-positive`
  - Available numbers closest to $-\infty$ or $+\infty$, respectively.
- `float-radix`, `float-digits`, `float-precision`
  - Radix, number of digits in that radix, or precision in that radix, respectively, of float $n$.
- `decode-float n`, `integer-decode-float n`
  - Return significand, exponent, and sign of float $n$.
- `scale-float n [i]`
  - With $n$'s radix $b$, return $nb^i$.
- `integer-decode-float n`
  - Return `integer-decode-float`.
- `float-radix n`, `float-digits n`, `float-precision n`
  - Radix, number of digits in that radix, or precision in that radix, respectively, of float $n$.
- `upgraded-complex-part-type foo (environment)`
  - Type of most specialized complex number able to hold parts of type `foo`.

2 Characters

The `standard-char` type comprises a-z, A-Z, 0-9, Newline, Space, and !?$*':;+-\_<>@()[]{}.

- `characterp char`
  - `T` if argument is of indicated type.
- `standard-char-p char`
  - `T` if `characterp`.
- `graphic-char-p character`
  - `T` if `character` is visible, alphabetic, or alphanumeric, respectively.
- `upper-case-p character`, `lower-case-p character`, `both-case-p character`
  - Return `T` if `character` is uppercase, lowercase, or able to be in another case, respectively.
- `digit-char-p character [radix]`
  - Return its weight if `character` is a digit, or `NIL` otherwise.
- `char= character`
  - `T` if all characters, or none, respectively, are equal.
- `char= greaterp character`
  - `T` if all characters, or none, respectively, are equal ignoring case.
- `char> character`
  - `T` if characters are monotonically decreasing, monotonically non-increasing, monotonically increasing, or monotonically non-decreasing, respectively.

3 Strings

Strings can be as well manipulated by array and sequence functions; see pages 10 and 12.

- `(string foo)`
  - `T` if `foo` is of indicated type.
- `(string= foo bar)`
  - Return `T` if subsequences of `foo` and `bar` are equal. Obey/ignore, respectively, case.
- `(string-float foo)`
  - Return string of length size.
- `(string x)`
  - Convert `x` (symbol, string, or character) into a string, a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.
- `(string-capitalization string)`
  - Convert string into a string with capitalized words, an all-uppercase string, or an all-lowercase string, respectively.
6 Sequences

6.1 Sequence Predicates

\(\{\text{every} \ (\text{not} \ \text{every})\}\)  
\(\text{test sequence+}\)  
\(\triangleright\) Return \(\text{NIL}\) or \(\top\), respectively, as soon as \(\text{test}\) on any set of corresponding elements of \(\text{sequences}\) returns \(\text{NIL}\).

\(\{\text{some} \ (\text{not} \ \text{any})\}\)  
\(\text{test sequence+}\)  
\(\triangleright\) Return value of \(\text{test}\) on \(\text{NIL}\), respectively, as soon as \(\text{test}\) on any set of corresponding elements of \(\text{sequences}\) returns \(\text{NIL}\).

\(\text{mismatch sequence-a sequence-b}\)  
\(\triangleright\) Return position in \(\text{sequence-a}\) where \(\text{sequence-a}\) and \(\text{sequence-b}\) begin to mismatch. Return \(\text{NIL}\) if they match entirely.

6.2 Sequence Functions

\(\{\text{make-sequence} \ \text{sequence-type size [initial-element foo]}\}\)  
\(\triangleright\) Make sequence of \(\text{sequence-type}\) with \(\text{size}\) elements.

\(\{\text{concatenate} \ \text{type sequence+}\}\)  
\(\triangleright\) Return concatenated sequence of \(\text{type}\).

\(\{\text{merge} \ \text{type} \ \text{sequence-a sequence-b test [key function]}\}\)  
\(\triangleright\) Return interleaved sequence of \(\text{type}\). Merged sequence will be sorted if both \(\text{sequence-a}\) and \(\text{sequence-b}\) are sorted.

\(\{\text{fill} \ \text{sequence} \ \text{foo test [start start-a]}\}\)  
\(\triangleright\) Return sequence after setting elements between \(\text{start}\) and \(\text{end}\) to \(\text{foo}\).

\(\{\text{length} \ \text{sequence}\}\)  
\(\triangleright\) Return length of \(\text{sequence}\) (being value of fill pointer if applicable).

\(\{\text{count} \ \text{foo sequence test [start start-a]}\}\)  
\(\triangleright\) Return number of elements in \(\text{sequence}\) which match \(\text{foo}\).

\(\{\text{count-if} \ (\text{not} \ \text{count-if})\}\)  
\(\text{test sequence [from-end end-a]}\)  
\(\triangleright\) Return number of elements in \(\text{sequence}\) which satisfy \(\text{test}\).

\(\{\text{elt} \ \text{sequence}\}\)  
\(\triangleright\) Return element of \(\text{sequence}\) pointed to by zero-indexed index. \text{setf}able.

\(\{\text{subset} \ \text{sequence} \ \text{start [end-a]}\}\)  
\(\triangleright\) Return subsection of \(\text{sequence}\) between \(\text{start}\) and \(\text{end}\). \text{setf}able.

\(\{\text{sort} \ (\text{stable-sort})\}\)  
\(\text{sequence test [key function]}\)  
\(\triangleright\) Return sequence sorted. Order of elements considered equal is not guaranteed/retained, respectively.

\(\{\text{reverse} \ \text{sequence}\}\)  
\(\triangleright\) Return \(\text{sequence}\) in reverse order.

\(\{\text{make-list} \ [\text{initial-element foo]}\}\)  
\(\triangleright\) Make list of \(\text{foo}\).
4.4 Trees

\[
\begin{align*}
\langle \text{tree-equal} \rangle & \text{ bar } \langle \text{test test} \rangle \\
\langle \text{tree-not} \rangle & \text{ test }
\end{align*}
\]

- Return \( T \) if trees \( \text{foo} \) and \( \text{bar} \) have same shape and leaves satisfying \( \text{test} \).

\[
\begin{align*}
\langle \text{subst} \rangle & \text{ new old tree } \\
\langle \text{subst} \rangle & \text{ new old tree }
\end{align*}
\]

- Make copy of \( \text{tree} \) with each subtree or leaf matching \( \text{old} \) replaced by \( \text{new} \).

\[
\begin{align*}
\langle \text{subst-if-not} \rangle & \text{ new test tree } \\
\langle \text{subst-if-not} \rangle & \text{ new test tree }
\end{align*}
\]

- Make copy of \( \text{tree} \) with each subtree or leaf satisfying \( \text{test} \) replaced by \( \text{new} \).

\[
\begin{align*}
\langle \text{sublis} \rangle & \text{ association-list tree } \\
\langle \text{sublis} \rangle & \text{ association-list tree }
\end{align*}
\]

- Make copy of \( \text{tree} \) with each subtree or leaf matching a key in \( \text{association-list} \) replaced by that key’s value.

\[
\langle \text{copy-tree} \rangle
\]

- Copy of \( \text{tree} \) with same shape and leaves.

4.5 Sets

\[
\begin{align*}
\langle \text{intersection} \rangle & \langle \text{set-difference} \rangle \langle \text{union} \rangle \\
\langle \text{set-exclusive-or} \rangle & \langle \text{intersection} \rangle \langle \text{set-difference} \rangle \\
\langle \text{nintersection} \rangle & \langle \text{nset-difference} \rangle \langle \text{nunion} \rangle \\
\langle \text{nset-exclusive-or} \rangle & \langle \text{nintersection} \rangle \langle \text{nset-difference} \rangle \\
\langle \text{key function} \rangle & \langle \text{test function} \rangle \langle \text{test-not function} \rangle
\end{align*}
\]

- Return \( a \cap b \), \( a \setminus b \), \( a \cup b \), or \( a \triangle b \), respectively, of lists \( a \) and \( b \).

5 Arrays

5.1 Predicates

\[
\begin{align*}
\langle \text{arrayp foo} \rangle \\
\langle \text{vectorp foo} \rangle \\
\langle \text{simple-vector-p foo} \rangle \\
\langle \text{bit-vector-p foo} \rangle
\end{align*}
\]

- Return \( T \) if \( \text{foo} \) is of indicated type.

\[
\begin{align*}
\langle \text{array-has-fill-pointer-p array} \rangle \\
\langle \text{array-in-bounds-p array \{subscripts\}} \rangle
\end{align*}
\]

- Return \( T \) if \( \text{array} \) is adjustable/has a fill pointer, respectively.

\[
\begin{align*}
\langle \text{make-array \{dimension-sizes\} \{adjustable bool\}} \rangle \\
\langle \text{adjust-array \{dimension-sizes\}} \langle \text{element-type type} \rangle \\
\langle \text{fill-pointer} \{\text{num} \{\text{bool}\}} \rangle \\
\langle \text{initial-element} \{\text{obj}\} \rangle \\
\langle \text{initial-contents} \{\text{tree-or-array}\} \rangle \\
\langle \text{displaced-to} \{\text{array} \{\text{displaced-index-offset\}} \rangle
\end{align*}
\]

- Return fresh, or adjust, respectively, vector or array.

\[
\begin{align*}
\langle \text{aref array \{subscripts\}} \rangle \\
\langle \text{row-major-aref array i} \rangle
\end{align*}
\]

- Return \( i \)th element of \( \text{array} \) in row-major order. setfable.

5.2 Array Functions

\[
\begin{align*}
\langle \text{array-row-major-index array \{subscripts\}} \rangle
\end{align*}
\]

- Index in row-major order of the element denoted by \( \text{subscripts} \).

\[
\begin{align*}
\langle \text{array-dimensions array} \rangle
\end{align*}
\]

- List containing the lengths of \( \text{array} \)'s dimensions.

\[
\begin{align*}
\langle \text{array-dimension array i} \rangle
\end{align*}
\]

- Length of \( i \)th dimension of \( \text{array} \).

\[
\begin{align*}
\langle \text{array-total-size array} \rangle
\end{align*}
\]

- Number of elements in \( \text{array} \).

\[
\begin{align*}
\langle \text{array-rank array} \rangle
\end{align*}
\]

- Number of dimensions of \( \text{array} \).

\[
\begin{align*}
\langle \text{array-displacement array \{offset\}} \rangle
\end{align*}
\]

- Target array and \( \text{offset} \).

\[
\begin{align*}
\langle \text{bit bit-array \{result\}} & \{\text{bit-array} \{\text{result-bit-array} \}} \rangle
\end{align*}
\]

- Return element of \( \text{bit-array} \) or of \( \text{simple-bit-array} \). setfable.

\[
\begin{align*}
\langle \text{bit-not \{result\}} & \{\text{bit-array} \{\text{result-bit-array} \}} \rangle
\end{align*}
\]

- Return result of bitwise negation of \( \text{bit-array} \). If result-\( \text{bit-array} \) is \( T \), put result in \( \text{bit-array} \); if it is \( \text{NIL} \), make a new array for result.

\[
\begin{align*}
\langle \text{bit-\&v} \{\text{bit-array} \{\text{bit-array-b} \}} \rangle
\end{align*}
\]

- Return result of bitwise logical operations (cf. operations of \( \text{boole} \), page 4) on \( \text{bit-array} \) and \( \text{bit-array-b} \). If result-\( \text{bit-array} \) is \( T \), put result in \( \text{bit-array} \); if it is \( \text{NIL} \), make a new array for result.

\[
\begin{align*}
\langle \text{array-rank-limit \rangle}
\end{align*}
\]

- Upper bound of array rank; \( \geq 8 \).

\[
\begin{align*}
\langle \text{array-dimension-limit \rangle}
\end{align*}
\]

- Upper bound of an array dimension; \( \geq 1024 \).

\[
\begin{align*}
\langle \text{array-total-size-limit \rangle}
\end{align*}
\]

- Upper bound of array size; \( \geq 1024 \).

5.3 Vector Functions

Vectors can as well be manipulated by sequence functions; see section 6.

\[
\begin{align*}
\langle \text{vector foo} \rangle
\end{align*}
\]

- Return fresh simple vector of \( \text{foo} \).

\[
\begin{align*}
\langle \text{svref vector i} \rangle
\end{align*}
\]

- Element \( i \) of simple vector. setfable.

\[
\begin{align*}
\langle \text{vector-push foo \{vector\}} \rangle
\end{align*}
\]

- Return \( \text{NIL} \) if vector’s fill pointer equals size of vector. Otherwise replace element of vector pointed to by \( \text{fill pointer} \) with \( \text{foo} \); then increment fill pointer.

\[
\begin{align*}
\langle \text{vector-push-extend foo \{vector\} \{num\}} \rangle
\end{align*}
\]

- Replace element of vector pointed to by \( \text{fill pointer} \) with \( \text{foo} \), then increment fill pointer. Extend vector’s size by \( \geq \text{num} \) if necessary.

\[
\begin{align*}
\langle \text{vector-pop \{vector\}} \rangle
\end{align*}
\]

- Return element of vector its fillpointer points to after decrementation.

\[
\begin{align*}
\langle \text{fill-pointer vector} \rangle
\end{align*}
\]

- Fill pointer of vector. setfable.
\(\text{fboundp \{foo \{setf foo\}\}}\)
\(\triangleright \) \(\text{NIL}\) if \(\text{foo}\) is a global function or macro.

9.2 Variables

\(\text{defconstant \{defparameter\}}\) \(\triangleright \) Assign value of form to global constant/dynamic variable \(\text{foo}\).

\(\text{defvar \text{foo \{form \{doc\}\}}}\)
\(\triangleright \) Unless bound already, assign value of form to dynamic variable \(\text{foo}\).

\(\text{set \{set \{psetf\}\}}\)
\(\triangleright \) Set places to primary values of forms. Return values of last form/\(\text{NIL}\): work sequentially/in parallel, respectively.

\(\text{set \{set \{psetq\}\}}\)
\(\triangleright \) Set symbols to primary values of forms. Return value of last form/\(\text{NIL}\): work sequentially/in parallel, respectively.

\(\text{set \{symbol \{foo\}\}}\)
\(\triangleright \) Set symbol's value cell to \(\text{foo}\). Deprecated.

\(\text{multiple-value-setq \text{vars \{form\}}}\)
\(\triangleright \) Sets elements of \(\text{vars}\) to the values of \(\text{form}\). Return \(\text{form}\) 's primary value.

\(\text{shift \{place \{+\} \{foo\}\}}\)
\(\triangleright \) Store value of \(\text{foo}\) in rightmost \(\text{place}\) shifting values of \(\text{place}\) left, returning first \(\text{place}\).

\(\text{rotate \{place\}}\)
\(\triangleright \) Rotate values of \(\text{place}\) left, old first becoming new last \(\text{place}\) 's value. Return \(\text{NIL}\).

\(\text{makunbound \{foo\}}\)
\(\triangleright \) Delete special variable \(\text{foo}\) if any.

\(\text{get \{symbol \{key \{default\}\}\}}}\)
\(\triangleright \) First entry key from property list stored in symbol/in place, respectively, or \(\text{default}\) if there is no key. \text{setf} able.

\(\text{get \{properties \{property-list \{keys\}\\}}}\)
\(\triangleright \) Return key and value of first entry from property-list matching a key from \(\text{keys}\), and tail of property-list starting with that key. Return \(\text{NIL}\), \(\text{NIL}\), and \(\text{NIL}\) if there was no matching key in property-list.

\(\text{remprop \{symbol \{key\}\}}}\)
\(\langle \text{remove \{first \{key\} \{from \{property \{list\}\} stored in \{symbol\}/in \{place\}, respectively\}}\rangle\)
\(\triangleright \) Return \(\text{NIL}\) if \(\text{key}\) was there, or \(\text{NIL}\) otherwise.

\(\text{remf \{place \{key\\}}}\)
\(\langle \text{remove \{first \{key\} \{from \{property \{list\}\} stored in \{symbol\}/in \{place\}, respectively\}}\rangle\)
\(\triangleright \) Return \(\text{NIL}\) if \(\text{key}\) was there, or \(\text{NIL}\) otherwise.

\(\text{progv \{values \{form\\}}}\)
\(\triangleright \) Evaluate \(\text{forms}\) with locally established dynamic bindings of \(\text{symbols}\) to \(\text{values}\) or \(\text{NIL}\). Return \(\text{values}\) of \(\text{forms}\).

\(\text{let \{\{let*\}\} \{name \{\{value\\}\}\} \{\{declaim\}\} \{\{form\\}\}}}\)
\(\triangleright \) Evaluate \(\text{forms}\) with \(\text{names}\) lexically bound (in parallel or sequentially, respectively) to \(\text{values}\). Return \(\text{values}\) of \(\text{forms}\).

\(\text{multiple-value-bind \{\{\\}\} \{values \{form\\}\} \{\{\\}\}\}}\)
\(\triangleright \) Evaluate \(\text{body-forms}\) with \(\text{vars}\) lexically bound to the return \(\text{values}\) of \(\text{values-form}\). Return \(\text{values}\) of \(\text{body-forms}\).
7 Hash Tables

The Loop Facility provides additional hash table-related functionality; see loop, page 21.

Key-value storage similar to hash tables can as well be achieved using association lists and property lists; see pages 9 and 16.

(\textbf{replace \textit{sequence-a} \textit{sequence-b}})  
\(\triangleright\) Replace elements of \textit{sequence-a} with elements of \textit{sequence-b}.

(\textbf{map \textit{type} \textit{function} \textit{sequence} +})  
\(\triangleright\) Apply \textit{function} successively to corresponding elements of the \textit{sequences}. Return values as a sequence of \textit{type}. If \textit{type} is \textit{NIL}, return \textit{NIL}.

(\textbf{map-into \textit{result-sequence} \textit{function} \textit{sequence} +})  
\(\triangleright\) Store into \textit{result-sequence} successively values of \textit{function} applied to corresponding elements of the \textit{sequences}.

(\textbf{reduce \textit{function} \textit{sequence}})  
\(\triangleright\) Starting with the first two elements of \textit{sequence}, apply \textit{function} successively to its last return value together with the next element of \textit{sequence}. Return last value of \textit{function}.

(\textbf{copy-seq \textit{sequence}})  
\(\triangleright\) Copy of \textit{sequence} with shared elements.

8 Structures

(\textbf{make-\textit{structure} \textit{sequence-a} \textit{sequence-b}})  
\(\triangleright\) Define \textit{structure} \textit{foo} together with functions \textit{MAKE-\textit{foo}}, \textit{COPY-\textit{foo}} and \textit{foo-P}, and \textit{setf}able accessors \textit{foo-SLOT}. Instances are of class \textit{foo}, or, if \textit{defstruct} option \textit{\textit{type}} is given, of the specified \textit{type}. They can be created by \{\textit{MAKE-\textit{foo} \textit{\{slot value\}}}\} or, if \textit{ord-\textit{\textit{lambda}}} (see page 17) is given, by \{\textit{maker arg} \textit{\{key value\}}\}. In the latter case, \textit{args} and \textit{\textit{key}}s correspond to the positional and keyword parameters defined in \textit{ord-\textit{\textit{lambda}}} whose \textit{vars} in turn correspond to \textit{slots}. \textit{print-object} method for an instance \textit{foo} calling \{\textit{o-printer} \textit{bar stream}\} or \{\textit{f-printer} \textit{bar stream print-level}\}, respectively.

(\textbf{copy-\textit{structure} \textit{structure}})  
\(\triangleright\) Return copy of \textit{structure} with shared slot values.

9 Control Structure

9.1 Predicates

(\textbf{eq \textit{foo} \textit{bar}})  
\(\triangleright\) \textit{foo} and \textit{bar} are identical.

(\textbf{equal \textit{foo} \textit{bar}})  
\(\triangleright\) \textit{foo} and \textit{bar} are \textit{equal}, or the same \textit{character}, or \textit{numbers} of the same \textit{type} and \textit{value}.

(\textbf{equalp \textit{foo} \textit{bar}})  
\(\triangleright\) \textit{foo} and \textit{bar} are \textit{\textit{equal}}, or are equivalent \textit{pathnames}, or are \textit{conses} with \textit{\textit{equal}} cars and cdrs, or are \textit{strings} or \textit{bit-vectors} with \textit{\textit{eq}} elements below their fill pointers.

(\textbf{equalp \textit{foo} \textit{bar}})  
\(\triangleright\) \textit{foo} and \textit{bar} are identical; or are the same \textit{character} ignoring case; or are \textit{numbers} of the same \textit{value} ignoring type; or are equivalent \textit{pathnames}; or are \textit{conses} or \textit{arrays} of the same \textit{shape} with \textit{\textit{equalp}} elements; or are \textit{hash-tables} of the same \textit{size} with the same \textit{\textit{test}} function, the same \textit{keys} in terms of \textit{\textit{test}} function, and \textit{\textit{equalp}} elements.

(\textbf{not \textit{foo}})  
\(\triangleright\) \textit{foo} is \textit{NIL}; \textit{NIL} otherwise.

(\textbf{boundp \textit{symbol}})  
\(\triangleright\) \textit{symbol} is a special \textit{variable}.

(\textbf{constantp \textit{foo} \textit{environment}})  
\(\triangleright\) \textit{foo} is a constant \textit{\textit{form}}.

(\textbf{functionp \textit{foo}})  
\(\triangleright\) \textit{foo} is of type \textit{\textit{function}}.
\begin{itemize}
  \item \texttt{case test \{ \{key\} \rightarrow \texttt{foo}, \{\texttt{otherwise}\} \rightarrow \texttt{bar} \}}
  \texttt{)}
  \rightarrow \text{Return the values of the first \texttt{foo}, one of whose keys is \texttt{eq test}. Return values of \texttt{bar} if there is no matching key.}

  \item \texttt{case test \{ \{key\} \rightarrow \texttt{foo}\}}
  \rightarrow \text{Return the values of the first \texttt{foo}, one of whose keys is \texttt{eq test}. Signal non-correctable/correctable type-error if there is no matching key.}

  \item \texttt{and form \texttt{m}}
  \rightarrow \text{Evaluate forms from left to right. Immediately return \texttt{NIL} if one form's value is \texttt{NIL}. Return values of last form otherwise.}

  \item \texttt{or form \texttt{n}}
  \rightarrow \text{Evaluate forms from left to right. Immediately return primary value of first non-\texttt{NIL}-evaluating form, or all values if last form is reached. Return \texttt{NIL} if no form returns \texttt{T}.}

  \item \texttt{progn form \texttt{n}}
  \rightarrow \text{Evaluate forms sequentially. Return values of last form.}

  \item \texttt{multiple-value-prog1 form-r form\texttt{r}}
  \rightarrow \text{Evaluate forms in order. Return values/primary value, respectively, of form-r.}

  \item \texttt{multiple-value-prog1 form-r form\texttt{r}}
  \rightarrow \text{Evaluate forms in order. Return values/primary value, respectively, of form-r.}

  \item \texttt{unwind-protect form \texttt{protected cleanup}}
  \rightarrow \text{Evaluate protected and then, no matter how control leaves \texttt{protected}, cleanup. Return values of \texttt{protected}.}

  \item \texttt{block name form\texttt{n}}
  \rightarrow \text{Evaluate forms in a lexical environment, and return their values unless interrupted by \texttt{return-from}.}

  \item \texttt{return-from form \texttt{result}}\texttt{result}
  \rightarrow \text{Have nearest enclosing \texttt{block} named \texttt{foo} named \texttt{NIL}, respectively, return with values of result.}

  \item \texttt{catch \texttt{tag form\texttt{n}}}
  \rightarrow \text{Evaluate forms in a lexical environment. \texttt{tags} (symbols or integers) have lexical scope and dynamic extent, and are targets for \texttt{go}. Return \texttt{NIL}.}

  \item \texttt{go \texttt{tag}}
  \rightarrow \text{Within the innermost possible enclosing \texttt{tagbody}, jump to a \texttt{tag}, \texttt{eq \texttt{tag}}.}

  \item \texttt{throw \texttt{tag form\texttt{n}}}
  \rightarrow \text{Have the nearest dynamically enclosing \texttt{catch} with a \texttt{tag}, \texttt{eq \texttt{tag}} return with the values of \texttt{form}.}

  \item \texttt{sleep \texttt{n}}
  \rightarrow \text{Wait \texttt{n} seconds; return \texttt{NIL}.}
\end{itemize}
9.4 Macros

Below, macro lambda list (macro-λ*) has the form of either

\[
[[\text{whole} \ \text{var}] \ E] \quad \text{[macro-λ]} \ E] \quad \text{[E]}
\]

\[
\&\text{optional} \quad \{\text{var} \ \text{[init]} \ \text{[supplied-p]}\} \quad \text{[E]}
\]

\[
\&\text{rest} \quad \text{[rest-var]} \quad \text{[macro-λ]} \ E
\]

\[
\&\text{key} \quad \{\text{var} \ \text{[key]} \ \text{[macro-λ]}\} \quad \text{[init]} \ \text{[supplied-p]}\} \quad \text{[E]}
\]

\[
\&\text{allow-other-keys} \quad \text{[&aux]} \quad \{\text{var} \ \text{[init]}\} \quad \text{[E]}
\]

or

\[
[[\text{whole} \ \text{var}] \ E] \quad \text{[macro-λ]} \ E] \quad \text{[E]} \quad \&\text{optional}
\]

\[
\text{[var} \ \text{[macro-λ]} \ \text{[init]} \ \text{[supplied-p]}\} \quad \text{[E]} \quad \text{rest-var}].
\]

One toplevel [E] may be replaced by \&\text{environment} \ var \ \supplied-p \ is \ T \ if \ there \ is \ a \ corresponding \ argument. \ init \ forms \ can \ refer \ to \ any \ init \ and \ supplied-p \ to \ their \ left.

\[
\&\text{defmacro} \quad \{\text{foo} \ \text{[setf foo]}\} \ \text{[macro-λ]} \ (\text{declare} \ \text{decl*}) \ \text{[doc form*]}
\]

\[
\&\text{define symbol macro} \ \text{foo form}
\]

\[
\&\text{macrolet} \quad \{\text{foo} \ \text{[macro-λ]} \ (\text{declare} \ \text{decl*}) \ \text{[macro-form*]}\} \ \text{[locally \ visible \ forms]}
\]

\[
\&\text{symbol macrolet} \quad \{\text{foo expansion-form*}\} \ \text{[declare decl*]} \ \text{form*}
\]

\[
\&\text{defun} \quad \{\text{update \ doc} \ \text{[self-λ]} \ \text{[s-var*]} \ \text{[declare decl*]} \ \text{[doc form*]}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]

\[
\&\text{call-arguments-limit} \quad \text{[lambda-parameters-limit]}
\]

\[
\&\text{multiple-values-limit}
\]
10 CLOS

10.1 Classes

(slot-exists-p foo bar) ⇒ T if foo has a slot bar.

(slot-boundp instance slot) ⇒ T if slot in instance is bound.

(declassify foo (superclass*) standard-object)

(slot reader reader*)
{writer (self writer*)}
{accessor accessor*}

(allocate instance class [instance]
{initarg :initarg-name*}
{initform form}
{type type}
{documentation slot-doc}
{metadata name standard-class}

(defclass foo (superclass* [standard-object])

(declare initarg-name*)

(:default-initargs (name value*)
{documentation class-doc}
{metadata name standard-class}

(find-class symbol [error# in environment])

⇒ Return class named symbol. setfable.

(make-instance class {initarg value*} other-keyarg*)

⇒ Make new instance of class.

(reinitialize-instance instance {initarg value*} other-keyarg*)

⇒ Change local slots of instance according to initargs by means of shared-initialize.

(slot-value foo slot) ⇒ Return value of slot in foo. setfable.

(slot-makunbound instance slot) ⇒ Make slot in instance unbound.

(with-slots {'slot var slot*)
(with-accessors {'var accessor*)

⇒ Return values of forms after evaluating them in a lexical environment with slots of instance visible as setfable slots or vars/with accessors of instance visible as setfable vars.

(class-name class)

{setf class-name new-name class)

⇒ Get/set name of class.

{class-of foo)

⇒ Class foo is a direct instance of.

(change-class instance new-class {initarg value*} other-keyarg*)

⇒ Change class of instance to new-class. Retain the status of any slots that are common between instance's original class and new-class. Initialize any newly added slots with the values of the corresponding initargs if any, or with the values of their initform forms if not.

(make-instances-obsolete class)

⇒ Update all existing instances of class using update-instance-for-redefined-class.

(update-instance instance
{update-instance-for-different-class previous current}
{initarg value*} other-keyarg*)

⇒ Set slots on behalf of make-instance of change-class by means of shared-initialize.

9.6 Iteration

{do |do*|} [var [start [step]]] (stop result*) (declare decl*)
{tag |tag*| form}
⇒ Evaluate tagbody-like body with var successively bound according to the values of the corresponding start and step forms. vars are bound in parallel/sequentially, respectively. Stop iteration when stop is T. Return values of result*. Implicitly, the whole form is a block named NIL.

(dotimes (var i (result i)) (declare decl*) {tag form*)
⇒ Evaluate tagbody-like body with var successively bound to integers from 0 to i - 1. Upon evaluation of result, var is i. Implicitly, the whole form is a block named NIL.

(dolist (var list (result i)) (declare decl*) {tag form*)
⇒ Evaluate tagbody-like body with var successively bound to the elements of list. Upon evaluation of result, var is NIL. Implicitly, the whole form is a block named NIL.

9.7 Loop Facility

(loop form*)
⇒ Simple Loop. If forms do not contain any atomic Loop Facility keywords, evaluate them forever in an implicit block named NIL.

(loop clause*)
⇒ Loop Facility. For Loop Facility keywords see below and Figure 1.

(named form)
⇒ Give loop's implicit block a name.

{with} [var-s {var-s*}] {d-type} [d-type*]
{and} [var-p {var-p*}] {d-type} [d-type*]
⇒ where destructuring type specifier d-type has the form
{fixnum} {float} [t NIL] {of-type type (type*)]
⇒ Initialize (possibly trees of) local variables var-s sequentially and var-p in parallel.

{foras} [var-s {var-s*}] {d-type*} [and] [var-p {var-p*}] {d-type*}
⇒ Begin of iteration control clauses. Initialize and step (possibly trees of) local variables var-s sequentially and var-p in parallel. Destructuring type specifier d-type as with with.

{upfrom} {downfrom} start
⇒ Start stepping with start
{upto} {downto} {top} {below} {above} form
⇒ Specify form as the end value for stepping.

{inon} list
⇒ Bind var to successive elements/tails, respectively, of list.

by \{step\} {function\}
⇒ Specify the (positive) decrement or increment or the function of one argument returning the next part of the list.
⇒ foo \then bar⇒ Bind var initially to foo and later to bar.

across vector
⇒ Bind var to successive elements of vector.
being \{the\} each
⇒ Iterate over a hash table or a package.

{hash-key \{hash-keys\} \{of in\} hash-table using \{hash-value \value\} value} \value\⇒ Bind var successively to the keys of hash-table; bind value to corresponding values.
Figure 1: Loop Facility, Overview.
(make-condition condition-type {('a:tiny-name value)})
  ➔ Return new instance of condition-type.

(signal condition-type {('a:tiny-name value)})
  ➔ Unless handled, signal as condition, warning or error, respectively, condition or a new instance of condition-type or, with format control and args (see page 36), simple-condition, simple-warning, or simple-error, respectively. From signal and warn, return NIL.

(error continue-control condition-type {('a:tiny-name value)})
  ➔ Unless handled, signal as correctable error condition or a new instance of condition-type or, with format control and args (see page 36), simple-condition, simple-warning, or simple-error, respectively. From error and setf, return NIL.

(quit)
  ➔ Return values of forms or, in case of errors, NIL and the condition.

:invoke-debugger condition
  ➔ Invoke debugger with control.

(declare 
  (var) (declare decl )
  (form)
  
  (defmethod general-method c-type c-arg* form)
  ➔ Return values of forms or, after evaluating them with control-functions, dynamically bound to their respective function types.

(defun with-simple-restart (restart) (control arg*) form)
  ➔ Return values of forms unless restart is called during their evaluation. In this case, describe restart using format control and args (see page 36) and return NIL and T.

(defun restart-case form (restart ord-\lambda*)
  (declare decl* restart-form\(\lambda\))
  ➔ Return values of form or, if during evaluation of form one of the dynamically established restarts is called, the values of its restart-form. A restart is visible under condition if (funcall test-function condition) returns T. If presented in the debugger, restarts are described by string or by #report-function (of a stream). A restart can be called by (invoke-restart restart arg*), where args match ord-\lambda*, or by (invoke-restart-interactively restart) where a list of the respective args is supplied by #arg-function. See page 17 for ord-\lambda*.

(update-instance-for-redefined-class new-instance added-slots discarded-slots discarded-slots-property-list (other-kegarg*))
  ➔ On behalf of make-instances obsolete and by means of share-initialize, set any initarg to its corresponding values; set any remaining added-slots to the values of their :initform forms. Not to be called by user.

(allocate-instance class initarg-name value other-kegarg*)
  ➔ Return uninitialized instance of class. Called by make-instance.

(allocate-instance class initarg-name value other-kegarg*)
  ➔ Fill the initarg-slots of instance with the corresponding values, and fill those initform-slots that are not initarg-slots with the values of their :initform forms.

(slot-missing class instance slot board)
  ➔ Called on attempted access to non-existing or unbound slot. Default methods signal error/unbound-slot, respectively. Not to be called by user.

10.2 Generic Functions

(next-method-p)
  ➔ T if enclosing method has a next method.

(defgeneric foo {setf foo}) (required-var* &optional (var*)),
  ➔ Define or modify generic function foo. Remove any methods previously defined by defgeneric. gf-class and the lambda parameters required-var* and var* must be compatible with existing methods. defmethod-args resemble those of defmethod. For c-type see section 10.3.

(ensure-generic-function foo {setf foo})
  ➔ Define or modify generic function foo. gf-class and lambda-list must be compatible with a pre-existing generic function or with existing methods, respectively. Changes to method-class do not propagate to existing methods. For c-type see section 10.3.
Define new method for generic function \textit{foo}. spec-vars specialize to either being of \texttt{class} or being \texttt{eq} \texttt{bar}, respectively. On invocation, vars and spec-vars of the new method act like parameters of a function with body \texttt{form}'. forms are enclosed in an implicit \texttt{block} \textit{foo}. Applicable qualifiers depend on the \texttt{method-combination} type; see section 10.3.

\begin{verbatim}
(add-method generic-function-method)
\end{verbatim}

\begin{verbatim}
(find-method generic-function-method arg)
\end{verbatim}

\begin{verbatim}
(compute-applicable-methods generic-function-method)
\end{verbatim}

\begin{verbatim}
(call-next-method arg)
\end{verbatim}

\begin{verbatim}
(no-applicable-method generic-function-method)
\end{verbatim}

\begin{verbatim}
(invalid-method-error method*)
\end{verbatim}

\begin{verbatim}
(method-combination-error control arg*)
\end{verbatim}

\begin{verbatim}
(next-method-method-combination-error)
\end{verbatim}

\begin{verbatim}
(function-keywords method)
\end{verbatim}

\begin{verbatim}
(method-qualifiers method)
\end{verbatim}

10.3 Method Combination Types

\textbf{standard}

\begin{itemize}
  \item \texttt{method-combination} \texttt{c-type}.
\end{itemize}

\begin{verbatim}
(method-combination-method arg)
\end{verbatim}

\begin{verbatim}
(no-method-method)
\end{verbatim}

\begin{verbatim}
(validate-keywords method)
\end{verbatim}

\begin{verbatim}
(validate-method-qualifiers method)
\end{verbatim}

11 Conditions and Errors

For standardized condition types cf. Figure 2 on page 31.

\begin{verbatim}
(defcondition foo (parent-type condition)
\end{verbatim}

\begin{verbatim}
(slot reader reader)
\end{verbatim}

\begin{verbatim}
(slot writer (setf writer))
\end{verbatim}

\begin{verbatim}
(slot accessor accessor)
\end{verbatim}

\begin{verbatim}
(slot instance instance)
\end{verbatim}

\begin{verbatim}
(slot class class)
\end{verbatim}

\begin{verbatim}
(slot documentation slot-doc)
\end{verbatim}

\begin{verbatim}
(slot report (report-function))
\end{verbatim}

\begin{verbatim}
(defmethod method)
\end{verbatim}

\begin{verbatim}
(make-method form)
\end{verbatim}

\begin{verbatim}
(next-method)
\end{verbatim}

\begin{verbatim}
(default-initargs (name value))
\end{verbatim}

\begin{verbatim}
(documentation condition-doc)
\end{verbatim}

\begin{verbatim}
(report string)
\end{verbatim}

\begin{verbatim}
(report-function)
\end{verbatim}

\textbf{Short Form.} Define new \texttt{method-combination} \texttt{c-type}. In a generic function using \texttt{c-type}, evaluate most specific \texttt{around} method supplying the values of the generic function. From within this method, \texttt{call-next-method} can call less specific \texttt{around} methods if there are any. If not, or if there are no \texttt{around} methods at all, return from the calling \texttt{call-next-method} or from the generic function, respectively, the values of (\texttt{operator (primary-method gen-arg)*}), \texttt{gen-arg} being the arguments of the generic function. The primary-methods are ordered \texttt{[most-specific-first] \texttt{most-specific-last}} (specified as \texttt{c-type} in \texttt{mdefgeneric}). Using \texttt{c-type} as the qualifier in \texttt{mdefmethod} makes the method primary.

\begin{verbatim}
(defgeneric foo (parent-type condition)
\end{verbatim}

\begin{verbatim}
(slot reader reader)
\end{verbatim}

\begin{verbatim}
(slot writer (setf writer))
\end{verbatim}

\begin{verbatim}
(slot accessor accessor)
\end{verbatim}

\begin{verbatim}
(slot instance instance)
\end{verbatim}

\begin{verbatim}
(slot class class)
\end{verbatim}

\begin{verbatim}
(slot documentation slot-doc)
\end{verbatim}

\begin{verbatim}
(slot report (report-function))
\end{verbatim}

\textbf{Call a to a generic function using \texttt{c-type} will be equivalent to a call to the forms returned by \texttt{body} with \texttt{ord-lambda} bound to \texttt{c-arg} (cf. \texttt{mdefformal}). Methods can be called via \texttt{call-method}. Lambda lists \texttt{(ord-lambda)} and \texttt{(method-combination-lambda)} according to \texttt{ord-lambda} on page 17, the latter enhanced by an optional \texttt{&whole} argument.

\textbf{Get standardized condition types cf. Figure 2 on page 31.}

\textbf{Condition Type Schema:}

\begin{verbatim}
(defcondition foo (parent-type condition)
\end{verbatim}

\begin{verbatim}
(slot reader reader)
\end{verbatim}

\begin{verbatim}
(slot writer (setf writer))
\end{verbatim}

\begin{verbatim}
(slot accessor accessor)
\end{verbatim}

\begin{verbatim}
(slot instance instance)
\end{verbatim}

\begin{verbatim}
(slot class class)
\end{verbatim}

\begin{verbatim}
(slot documentation documentation)
\end{verbatim}

\begin{verbatim}
(slot report (report-function))
\end{verbatim}

\textbf{Define, as a subtype of parent-types, condition type \texttt{foo}.}

\textbf{In a new condition, a \texttt{slot}'s value defaults to \texttt{form} unless set via \texttt{initarg-name}; it is readable via (\texttt{reader i}) or (\texttt{accessor i}), and writable via (\texttt{writer value}) or (\texttt{setf accessor i}) value). With \texttt{allocation class}, \texttt{slot} is shared by all conditions of type \texttt{foo}. A condition is reported by \texttt{string} or by \texttt{report-function} of arguments condition and stream.
13 Input/Output

13.1 Predicates

(streamp foo) ▷ T if foo is of indicated type.
(pathnamep foo) ▷ T if foo is of indicated type.
(readatable foo) ▷ T if foo is of indicated type.
(input-stream-p stream) ▷ Return T if stream is for input, for output, interactive, or open, respectively.
(output-stream-p stream) ▷ Return T if stream is for input, for output, interactive, or open, respectively.
(interactive-stream-p stream) ▷ Return T if stream is for input, for output, interactive, or open, respectively.
(open-stream-p stream) ▷ Return T if stream is for input, for output, interactive, or open, respectively.
(pathname-match-p path wildcard) ▷ T if path matches wildcard.
(wild-pathname-p path [:host] [:device] [:directory] :name] [:type] version [NIL]) ▷ Return T if indicated component in path is wildcard. (NIL indicates any component.)

13.2 Reader

(y-or-n-p) ▷ Ask user a question and return T or NIL depending on their answer. See page 36, format. For control and args.
(yes-or-no-p) ▷ Ask user a question and return T or NIL depending on their answer. See page 36, format. For control and args.
(read (read-preserving-whitespace) (stream) [standard-input] [eof-error nil] [eof-error (recursive nil)]) ▷ Read printed representation of object.
(read-from-string string) [eof-error] [eof-value nil] [start start] [end end] [preserve-whitespace true] ▷ Return object read from string and zero-indexed position of next character.
(read-delimited-list char stream) [standard-input] [recursive true] ▷ Continue reading until encountering char. Return list of objects read. Signal error if no char is found in stream.
(read-char stream) [standard-input] [eof-error nil] [eof-value nil] [recursive nil] ▷ Return next character from stream.
(read-char-no-hang stream) [standard-input] [eof-error nil] [eof-value nil] [recursive nil] ▷ Next character from stream or NIL if none is available.
(peek-char mode stream) [standard-input] [eof-error nil] [eof-value nil] [recursive nil] ▷ Next, or if mode is 1, next non-whitespace character, or if mode is a character, next instance of it, from stream without removing it there.
(unread-char character stream) ▷ Put last read character back into stream; return NIL.
(read-byte stream) [eof-error nil] [eof-value nil] ▷ Read next byte from binary stream.
(read-line stream) [standard-input] [eof-error nil] [eof-value nil] [recursive true] ▷ Return a line of text from stream and T if line has been ended by end of file.

13.3 Debugging

(restart-bind ( (restart) restart-function) (interactive-function arg-function) (report-function report-function) (test-function test-function)) ▷ Return values of forms evaluated with dynamically established restarts whose restart-functions should perform a non-local transfer of control. A restart is visible under condition if (test-function condition) returns T. If presented in the debugger, restarts are described by restart-function (of a stream). A restart can be called by (invoke-restart restart arg*), where args must be suitable for the corresponding restart-function, or by (invoke-restart-interactively restart) where a list of the respective args is supplied by arg-function.

(invoke-restart restart arg*) ▷ Call function associated with restart with arguments given or prompted for, respectively. If restart function returns, return its values.
(find-restart) ▷ Return innermost restart name, or a list of all restarts, respectively, of those either associated with condition or un-associated at all; or, without condition, out of all restarts. Return NIL if search is unsuccessful.
(restart-name restart) ▷ Name of restart.

(abort) ▷ Name of condition.
(muffle-warning) ▷ Name of condition.
(continue) ▷ Name of condition.
(store-value value) ▷ Name of condition.
(use-value value) ▷ Name of condition.

(with-condition-restarts condition restarts form) ▷ Evaluate forms with restarts dynamically associated with condition. Return values of forms.
(arithmetic-error-operation condition) ▷ Name of condition.
(arithmetic-error-operands condition) ▷ Name of condition.
(cell-error-name condition) ▷ Name of condition.
(unbound-slot-instance condition) ▷ Name of condition.
/print-not-readable-object condition ▷ Name of condition.

(package-error-package condition) ▷ Name of condition.
(file-error-pathname condition) ▷ Name of condition.
(stream-error-stream condition) ▷ Name of condition.
(type-error-datum condition) ▷ Name of condition.
(type-error-expected-type condition) ▷ Name of condition.
(simple-condition-format-control condition) ▷ Name of condition.
(simple-condition-format-arguments condition) ▷ Name of condition.
*break-on-signals* ▷ Name of condition.
### 12 Types and Classes

For any class, there is always a corresponding type of the same name.

\((\text{typep} \ foo \ \text{type} \ \text{environment})\) \(\Rightarrow\) \(T\) if \(foo\) is of type.

\((\text{subtypep} \ \text{type-a} \ \text{type-b} \ \text{environment})\)

\(\Rightarrow\) Return \(T\) if \(\text{type-a}\) is a recognizable subtype of \(\text{type-b}\), and \(NIL\) if the relationship could not be determined.

\((\text{the} \ \text{type form})\) \(\Rightarrow\) Declare values of \(\text{form}\) to be of type.

\((\text{ coerce} \ \text{object type})\) \(\Rightarrow\) Coerce \(\text{object}\) into type.

\((\text{etypecase} \ foo \ (\text{type a-form}^b)^* \ [(\text{otherwise} \ b-form^b)])\)

\(\Rightarrow\) Return values of the first \(a\)-form\(^b\) whose type is \(foo\) of. Return values of \(b\)-forms if no type matches.

\((\text{the} \ \text{type form})\) \(\Rightarrow\) Declare values of \(\text{form}\) to be of type.

\((\text{ coerce} \ \text{object type})\) \(\Rightarrow\) Coerce \(\text{object}\) into type.

\((\text{ stream-element-type} \ \text{stream})\) \(\Rightarrow\) Type of stream objects.

\((\text{ array-element-type} \ \text{array})\) \(\Rightarrow\) Element type \(\text{array}\) can hold.

\((\text{ upgraded-array-element-type} \ \text{type} \ \text{environment})\)

\(\Rightarrow\) Element type of most specialized array capable of holding elements of \text{type}.

\((\text{deftype} \ foo \ (\text{ macro-λ}^*) \ (\text{declare doc}^*) \ [\text{doc form}^b])\)

\(\Rightarrow\) Define type \(foo\) which when referenced as (\(foo \ \text{doc}^*\)) (or as \(foo\) if \(\text{macro-λ}\) doesn't contain any required parameters) applies expanded \text{forms} to \text{args} returning the new type. For \(\text{macro-λ}\) see page 18 but with default value of \(\text{block}\) instead of \text{NIL}. \text{forms} are enclosed in an implicit \text{block} named \(foo\).

\((\text{ eql} \ foo)\)

\((\text{ member} \ foo^*)\) \(\Rightarrow\) Specifier for a type comprising \(foo\) or \(foos\).

\((\text{satisfies} \ \text{predicate})\)

\(\Rightarrow\) Type specifier for all objects satisfying \(\text{predicate}\).

\((\text{ mod} \ n)\)

\(\Rightarrow\) Type specifier for all non-negative integers \(< n\).

\((\text{ not} \ \text{type})\)

\(\Rightarrow\) Complement of type.

\((\text{ and} \ \text{type}^a)\)

\(\Rightarrow\) Type specifier for intersection of types.

\((\text{ or} \ \text{type}^a)\)

\(\Rightarrow\) Type specifier for union of types.

\((\text{values} \ \text{type}^a \ \text{optional} \ \text{type}^a \ \text{[rest other-args]])\)

\(\Rightarrow\) Type specifier for multiple values.

\(\ast\)

\(\Rightarrow\) As a type argument (cf. Figure 2): no restriction.

---

**Figure 2**: Precedence Order of System Classes ( ), Classes ( ), Types ( ), and Condition Types ( ).

Every type is also a supertype of NIL, the empty type.
(prin1-newline :linear :fill :mandatory)
  ➡ Print a conditional newline if stream is a pretty printing
  stream. Return NIL.

*(print-array* ➡ If T, print arrays ,readably.

*(print-base* ➡ Radix for printing rationals, from 2 to 36.

*(print-case* ➡ Print symbol names all uppercase (:upcase), all lowercase
  (:downcase), capitalized (:capitalize).

*(print-circle* ➡ If T, avoid indefinite recursion while printing circular
  structure.

*(print-escape* ➡ If NIL, do not print escape characters and package
  prefixes.

*(print-gensym* ➡ If T, print #: before uninterned symbols.

*(print-length* ➡ If T, print rationals with a radix indicator.

*(print-readably* ➡ If T, print ,readably or signal error print-not-readable.

*(print-right-margin* ➡ Right margin width in ems while pretty-printing.

(set-pprint-dispatch type function [priority [table]] ➡ Install entry comprising
  function of arguments stream and object to print; and priority as type into table. If
  function is NIL, remove type from table. Return NIL.

(printf print-dispatch foo [table] ➡ Return highest priority function associated with type of
  foo and T if there was a matching type specifier in table.

(copy-pprint-dispatch [table] ➡ Return copy of table or, if table is NIL, initial value of
  *printf-dispatch*.

*printf-dispatch* ➡ Current pretty print dispatch table.

13.5 Format

(formatter control) ➡ Return function of stream and arg* applying format to
  stream, control, and arg* returning NIL or any excess args.

(format [\#fill out-stream out-stream] control arg*) ➡ Output string control which may contain \- directives possibly taking some args. Alternatively, control can be a
  function returned by formatter which is then applied to
  out-stream and arg*. Output to out-stream, out-stream or, if first argument is T, to
  *standard-output*. Return NIL. If first argument is NIL, return formatted output.

(\read-sequence sequence stream [start start] [end end] ➡ Replace elements of sequence between start and end with elements from binary or character stream. Return index of
  sequence’s first unmodified element.

(\readable-case readable) ➡ Case sensitivity attenuate (one of :upcase, :downcase,
  :preserve, :invert) of readable. setfable.

(copy-\readable* ➡ Return copy of from-\readable*.

(set-synt\from-char-to-char ➡ Copy syntax of from-char to to-char. Return T.

*readables* ➡ Current readable.

*read-base* ➡ Radix for reading integers and ratios.

*read-default-\float-format ➡ Floating point format to use when not indicated in the
  number read.

*read-suppress* ➡ If T, reader is syntactically more tolerant.

(set-macro-character char function [non-term-\readable*] ➡ Make char a macro character associated with function
  of stream and char. Return T.

(get-macro-character char [non-term-\readable*] ➡ Reader macro function associated with char, and T if
  char is a non-terminating macro character.

(make-dispatch-macro-character char [non-term-\readable*] ➡ Make char a dispatching macro character. Return T.

(set-dispatch-macro-character char sub-char function [\readable*] ➡ Make function of stream, n, sub-char a dispatch function
  of char followed by n, followed by sub-char. Return T.

(get-dispatch-macro-character char sub-char [\readable*] ➡ Dispatch function associated with char followed by
  sub-char.

13.3 Character Syntax

#: multi-line-comment* |#
  one-line-comment* ➡ Comments. There are stylistic conventions:

;;; title ➡ Short title for a block of code.

;;; intro ➡ Description before a block of code.

;; state ➡ State of program or of following code.

; explanation ➡ Regarding line on which it appears.

(foo*[ . bar]) ➡ List of foos with the terminating cdr bar.

" ➡ Begin and end of a string.

\foo ➡ (\quote foo); foo unevaluated.

\[\foo [\[baz] [\[quux] [\[bing]]]] ➡ Backquote. \quote foo and bing; evaluate bar and splice
  the lists baz and quux into their elements. When nested, outermost commas inside the innermost backquote
  expression belong to this backquote.

#\c ➡ (\character "c"), the character c.

#\2n; #\0n; #\nx; #\r ➡ Integer of radix 2, 8, 10, 16, or \r; 2 ≤ r ≤ 36.
n/d \( \mapsto \) The ratio \( \frac{n}{d} \).

\[
\{m.n|n|[(S|F|D|E)|\overline{E}]|\overline{m}n|([m.|n|]S|F|D|E)\}\n\]

- \( m.n \cdot 10^d \) as short-float, single-float, double-float, long-float, or the type from \texttt{read-default-float-format}.  

#(a b) \( \mapsto \) (\texttt{complex a b}), the complex number \( a + bi \).  

#'foo \( \mapsto \) (\texttt{function foo}); the function named \texttt{foo}.  

#:nAsequence \( \mapsto \) n-dimensional array.  

#:n\(\text{foo}^*\) \( \mapsto \) Vector of some (or none) \texttt{foo} filled with last \texttt{foo} if necessary.  

#:n\(\text{b}^*\) \( \mapsto \) Bit vector of some (or none) \texttt{b} filled with last \texttt{b} if necessary.  

#:S\{\text{type} \{ \text{slot value}\}^*\} \( \mapsto \) Structure of type.  

#:Pstring \( \mapsto \) A pathname.  

#:foo \( \mapsto \) Uninterpreted symbol \texttt{foo}.  

#:form \( \mapsto \) Read-time value of form.  

#:read-eval\textbackslash paul \( \mapsto \) If NIL, a \texttt{reader-error} is signalled at \#.  

#:integer\texttt{=} foo \( \mapsto \) Give \texttt{foo} the label integer.  

#:integer\# \( \mapsto \) Object labelled integer.  

#:< \( \mapsto \) Have the reader signal \texttt{reader-error}.  

#:feature when-feature  

#:feature unless-feature \( \mapsto \) Means when-feature if feature is \( \top \); means unless-feature if feature is \texttt{NIL}. Feature is a symbol from \texttt{.features}, or \((\texttt{and}|\texttt{or})\) feature\texttt{*}, or \texttt{not} feature.  

#:features \( \mapsto \) List of symbols denoting implementation-dependent features.  

\texttt{[c*]:|c} \( \mapsto \) Treat arbitrary character(s) \texttt{c} as alphabetic preserving case.  

### 13.4 Printer

- \(\texttt{print}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \} \\
&\{\text{print-1}\}\{\text{to-string} \}
\end{align*}
\]

- \(\texttt{print}\) \( \mapsto \) Print \texttt{foo} to stream \texttt{readably}, \texttt{readably} between a newline and a space, \texttt{readably} after a newline, or human-readably without any extra characters, respectively. \texttt{prinl}, \texttt{print} and \texttt{princ} return \texttt{foo}.  

- \(\texttt{print-1}\)

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{print\texttt{-to-string} foo}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{print\texttt{-object object stream} \texttt{-}}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{print\texttt{-unreadable-object (foo stream \{:type \texttt{null} \}}\texttt{ form\textbackslash n})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{terpri (stream \texttt{-standard-output})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{fresh-line (stream \texttt{-standard-output})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

### 13.5 Formatting

- \(\texttt{write-char char (stream \texttt{-standard-output})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write-string string (stream \texttt{-standard-output})\{[:start \texttt{start\texttt{-}}]}\{[:end \texttt{end\texttt{-}}]\}\}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write-byte byte stream}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write-sequence sequence stream\{[:start \texttt{start\texttt{-}}]}\{[:end \texttt{end\texttt{-}}]\}\}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write\texttt{-to-string} foo}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write\texttt{-object object stream} \texttt{-}}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{write\texttt{-unreadable-object \{foo stream \{[:type \texttt{null} \}}\texttt{ form\textbackslash n})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{terpri (stream \texttt{-standard-output})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]

- \(\texttt{fresh-line (stream \texttt{-standard-output})}\)  

\[
\begin{align*}
&\{\text{prinl} \} \\
&\{\text{print} \} \\
&\{\text{princ} \}
\end{align*}
\]
Common Lisp Quick Reference

\( \text{(close stream \[abort bool\])} \)
\( \triangleright \) Close stream. Return \( T \) if stream had been open. If \( \text{abort} \) is \( T \), delete associated file.

\( \text{(with-open-file (stream path open-args) (declare decl*) form*)} \)
\( \triangleright \) Use \( \text{open} \) with \( \text{open-args} \) to temporarily create stream to path; return values of forms.

\( \text{(with-open-stream (foo stream) (declare decl*) form*)} \)
\( \triangleright \) Evaluate forms with \( \text{foo} \) locally bound to stream. Return values of forms.

\( \text{(with-input-from-string (foo stream) \( \left( \text{index index} \right) \) (declare decl*) form*)} \)
\( \triangleright \) Evaluate forms with \( \text{foo} \) locally bound to input string-stream from string. Return values of forms; store next reading position into index.

\( \text{(with-output-to-string (foo stream) \( \left( \text{type type} \right) \) (declare decl*) form*)} \)
\( \triangleright \) Evaluate forms with \( \text{foo} \) locally bound to an output string-stream. Append output to string and return values of forms if string is given. Return string containing output otherwise.

\( \text{(stream-external-format stream)} \)
\( \triangleright \) External file format designer.

\( \text{*terminal-io*} \)
\( \triangleright \) Bidirectional stream to user terminal.

\( \text{*standard-input*} \)
\( \text{*standard-output*} \)
\( \text{*error-output*} \)
\( \triangleright \) Standard input stream, standard output stream, or standard error output stream, respectively.

\( \text{*debug-io*} \)
\( \text{*query-io*} \)
\( \triangleright \) Bidirectional streams for debugging and user interaction.

### 13.7 Pathnames and Files

\( \text{(makepathname \( \left( \right) \)} \)
\( \triangleright \) Construct a logical pathname if there is a logical pathname translation for \( \text{host} \), otherwise construct a physical pathname. For \( \text{case} \) local, leave case of components unchanged. For \( \text{case} \) common, leave mixed-case components unchanged; convert all-uppercase components into local customary case; do the opposite with all-lowercase components.

\( \text{(pathname-host \( \text{pathname} \)} \)
\( \text{(pathname-device \( \text{pathname} \)} \)
\( \text{(pathname-directory \( \text{pathname} \)} \)
\( \text{(pathname-name \( \text{pathname} \)} \)
\( \text{(pathname-type \( \text{pathname} \)} \)
\( \text{(pathname-version \( \text{pathname} \)} \)
\( \triangleright \) Return pathname component.

\( \text{- min-col} \left( [\text{col-width}] \left( \text{[min-pad]} \left( [\text{pad-char}] \right) \right) \right) \)
\( \left[ \text{[nil-text]} \left( \text{[sparrow]} \left( \text{[width]} \right) \right) \right] \left( \text{[text]} \right) \)
\( \triangleright \) Minimum column. Print argument of any type for consumption by humans; by the reader, respectively. With \( \text{nil} \), print \( \text{NIL} \) as \( \ell \) rather than \( \text{NIL} \). With \( \text{[nil-text]} \), add \( \text{pad-chars} \) on the left rather than on the right.

\( \text{- radix} \left( [\text{width}] \left( [\text{pad-char}] \left( \text{[comma-char]} \right) \right) \right) \)
\( \left( \text{[comma-intervel]} \right) \left[ \text{[none]} \right] \left( \text{[0]} \right) \left( \text{[R]} \right) \)
\( \triangleright \) Radix. (With one or more prefix arguments.) Print argument as number; with \( \ell \), group digits comma-interval each; with \( \text{[0]} \), always prepend a sign.

\( \text{-R} \left( \text{-R} \left( \text{[0]} \right) \right) \)
\( \triangleright \) Roman. Take argument as number and print it as English cardinal number, as English ordinal number, as Roman numeral, or as old Roman numeral, respectively.

\( \text{- width} \left( [\text{pad-char}] \left( \text{[comma-char]} \right) \right) \)
\( \left( \text{[comma-intervel]} \right) \left[ \text{[none]} \right] \left( \text{[0]} \right) \left( \text{[R]} \right) \)
\( \triangleright \) Decimal/Binary/Octal/Hexadecimal. Print integer argument as number. With \( \ell \), group digits comma-interval each; with \( \text{[0]} \), always prepend a sign.

\( \text{- width} \left( \text{[dec-digits]} \left( \text{[shift]} \left( \text{[overflow-char]} \right) \right) \right) \)
\( \left( \text{[pad-char]} \right) \left( \text{[width]} \right) \left( \text{[comma-char]} \right) \left[ \text{[none]} \right] \left( \text{[0]} \right) \left( \text{[F]} \right) \)
\( \triangleright \) Fixed-Format Floating-Point. With \( \ell \), always prepend a sign.

\( \text{- width} \left( \text{[dec-digits]} \left( \text{[exp-digits]} \left( \text{[scale-factor]} \right) \right) \right) \)
\( \left( \text{[overflow-char]} \left( \text{[pad-char]} \right) \right) \left( \text{[width]} \right) \left( \text{[comma-char]} \right) \left[ \text{[none]} \right] \left( \text{[0]} \right) \left( \text{[E]} \left( \text{[G]} \right) \right) \)
\( \triangleright \) Exponential/General Floating-Point. Print argument as floating-point number with \( \text{dec-digits} \) after decimal point and \( \text{exp-digits} \) in the signed exponent. With \( \text{-G} \), choose either \( \text{-E} \) or \( \text{-F} \). With \( \text{[0]} \), always prepend a sign.

\( \text{- dec-digits} \left( \text{[int-digits]} \left( \text{[width]} \right) \left( \text{[pad-char]} \right) \right) \left[ \text{[none]} \right] \left( \text{[0]} \right) \left( \text{[5]} \right) \)
\( \triangleright \) Monetary Floating-Point. Print argument as fixed-format floating-point number. With \( \ell \), put sign before any padding; with \( \text{[0]} \), always prepend a sign.

\( \text{-C} \left( \text{-C} \left( \text{[0]} \right) \right) \)
\( \triangleright \) Character. Print, spell out, print in \#\text{\textbackslash} syntax, or tell how to type, respectively, argument as (possibly non-printing) character.

\( \text{- text} \left( \text{-text} \left( \text{[text]} \right) \right) \left( \text{[text]} \right) \left( \text{[text]} \right) \)
\( \triangleright \) Case-Conversion. Convert text to lowercase, convert first letter of each word to uppercase, capitalize first word and convert the rest to lowercase, or convert to uppercase, respectively.

\( \text{-P} \left( \text{-P} \left( \text{[0]} \right) \right) \)
\( \triangleright \) Plural. If argument \text{eq1} \ print nothing, otherwise print \text{n} do the same for the previous argument; if argument \text{eq1} \ print \text{y}, otherwise print \text{es} \ do the same for the previous argument, respectively.

\( \text{-r} \left( \text{-r} \right) \)
\( \triangleright \) Newline. Print \( n \) newlines.

\( \text{-f} \left( \text{\textbackslash} \right) \)
\( \triangleright \) Fresh-Line. Print \( n - 1 \) newlines if output stream is at the beginning of a line, or \( n \) newlines otherwise.

\( \text{-t} \left( \text{\textbackslash} \right) \text{\textbackslash} \)
\( \triangleright \) Conditional Newline. Print a newline like \text{print-newline} with argument \text{:line}, \text{:fill}, \text{:miser}, or \text{:mandatory}, respectively.

\( \text{-t} \left( \text{\textbackslash} \right) \)
\( \triangleright \) Ignored Newline. Ignore newline, or whitespace following newline, or both, respectively.

\( \text{-l} \left( \text{l} \right) \)
\( \triangleright \) Page. Print \( n \) page separators.

\( \text{-w} \left( \text{\textbackslash} \right) \)
\( \triangleright \) Tilde. Print \( n \) tildes.

\( \text{-min-col} \left( [\text{col-width}] \left( [\text{min-pad}] \left( [\text{pad-char}] \right) \right) \right) \)
\( \left[ \text{[nil-text]} \left( \text{[sparrow]} \left( \text{[width]} \right) \right) \right] \left( \text{[text]} \right) \)
\( \triangleright \) Minimum column. Print argument of any type for consumption by humans; by the reader, respectively. With \( \text{nil} \), print \( \text{NIL} \) as \( \ell \) rather than \( \text{NIL} \). With \( \text{[nil-text]} \), add \( \text{pad-chars} \) on the left rather than on the right.
Justification. Justify text produced by `text` in a field of at least non-col columns. With `,` right justify; with `,`, left justify. If this would leave less than spare characters on the current line, output `nil`-text first.

- `[i] {}@ `<(prefix - [:per-line-prefix - @:i]; suffix - [:@i]` body - [:;]
  ▶ Logical Block. Act like `print-logical-block` using `body` as `:format` control string on the elements of the list argument or, with `{@i}, on the remaining arguments, which are extracted by `print-pop`. With `, prefix and suffix default to `{` and `}`. When closed by `-:@i`, spaces in `body` are replaced with conditional newlines.

- `i` [-] [:i]`
  ▶ Indent. Set indentation to `n` relative to leftmost/to current position.

- `[e] [i] [:] {T]`
  ▶ Tabulate. Move cursor forward to column number `c+k`, `k` ≥ 0 being as small as possible. With `, calculate column numbers relative to the immediately enclosing section. With `{`, move to column number `c0 + c + k` where `c0` is the current position.

- `[e] [i] [:] {n}`
  ▶ Go-to. Jump `n` arguments forward, or backward, or to argument `n`.

- `:limit [:] {text -}
  ▶ Iteration. Use `text` repeatedly, up to `limit`, as control string for the elements of the list argument or (with `{`) for the remaining arguments. With `{` or `{`, list elements or remaining arguments should be lists of which a new one is used at each iteration step.

- `[x] [y] [z]`
  ▶ Escape Upward. Leave immediately `< - ->`, `<- < - ->`, `- <`, or the entire `:format` operation. With one to three prefixes, act only if `x = 0`, `x = y`, or `x ≤ y ≤ z`, respectively.

- `[x] [:] {text -}[y] text -[: ;[; default] -]
  ▶ Conditional Expression. Use the zero-indexed argument (or `ith` if given) `text` as a `:format` control sub-clause. With `,`, use the first `text` if the argument value is `NIL` or the second `text` if it is `T`. With `{`, do nothing for an argument value of `NIL`. Use the only `text` and leave the argument to be read again if it is `T`.

- `?-[:@?]`
  ▶ Recursive Processing. Process two arguments as control string and argument list, or take one argument as control string and use then the rest of the original arguments.

- `[prefix {prefix `-[:package -[:association function/]
  ▶ Call Function. Call all-uppercase `package.function` with the arguments stream, format-argument, colon-p, at-sign-p and prefixes for printing format-argument.

- `[i] {W`
  ▶ Write. Print argument of any type obeying every printer control variable. With `,`, pretty-print. With `{`, print without limits on length or depth.

- `{V}`
  ▶ In place of the comma-separated prefix parameters: use next argument or number of remaining unprocessed arguments, respectively.

---

### 13.6 Streams

- `[:direction {input output} :io)`
- `[:probe type]:element-type`{default} character
  - `[:open path]`
    - `:if-exists` synonym
    - `:if-not-exist` synonym
  - `:create error` N
  - `:error error` RL for direction `probe`; `create` error otherwise
- `[:external-format form] stream`

- `(:make-concatenated-stream input-stream+)`
- `(:make-broadcast-stream output-stream+)`
- `(:make-two-way-stream input-stream-part output-stream-part)`
- `(:make-echo-stream from-input-stream to-output-stream)`
- `(:make-synonym-stream variable-bound-to-stream)`
  ▶ Return stream of indicated type.

- `(:make-string-input-stream string [start [end]]])`
  ▶ Return a `string-stream` supplying the characters from `string`.

- `(:make-string-output-stream :element-type synonym)`
  ▶ Return a `string-stream` accepting characters (available via `get-output-stream-string`).

- `(:concatenated-stream-streams concatenated-stream)`
- `(:broadcast-stream-streams broadcast-stream)`
  ▶ Return list of streams `concatenated-stream` still has to read from `broadcast-stream` is broadcasting to.

- `(:two-way-stream-input-stream two-way-stream)`
- `(:two-way-stream-output-stream two-way-stream)`
- `(:echo-stream-input-stream echo-stream)`
- `(:echo-stream-output-stream echo-stream)`
  ▶ Return source stream or sink stream of `two-way-stream`.

- `(:synonym-stream-symbol synonym-stream)`
  ▶ Return `symbol` of `synonym-stream`.

- `(:get-output-stream-string string-stream)`
  ▶ Clear and return as a `string` characters on `string-stream`.

- `(:file-position stream {start [end] position})`
  ▶ Return position within stream, or set it to `position` and return `T` on success.

- `(:file-string-length stream foo)`
  ▶ Length `foo` would have in stream.

- `(:listen [stream] standard-input)`
  ▶ `T` if there is a character in input stream.

- `(:clear-input [stream] standard-input)`
  ▶ Clear input from stream, return `NIL`.

- `(:clear-output [stream] standard-output)`
  ▶ End output to `stream` and return `NIL` immediately, after initiating flushing of buffers, or after flushing of buffers, respectively.
14.4 Standard Packages

common-lisp

▷ Exports the defined names of Common Lisp except for those in the keyword package.

common-lisp-user|cl-user

▷ Current package after startup; uses package common-lisp.

keyword

▷ Contains symbols which are defined to be of type keyword.

15 Compiler

15.1 Predicates

(special-operator-p foo) ▷ T if foo is a special operator.

(compiled-function-p foo) ▷ T if foo is of type compiled-function.

15.2 Compilation

(compile (NIL definition

(setf name definition))

▷ Return compiled function or replace name’s function definition with the compiled function. Return T in case of warnings or errors, and NIL in case of warnings or errors excluding style-warnings.

(compile-file file

(verbatim (compile-varname)

(print file (compile-print)

(output-file out-path

(verbatim (compile-verbatim)

(print (verbatim (compile-print)

(verbatim (compile-verbatim)

(setf name definition)))

▷ Write compiled contents of file to out-path. Return true output path or NIL, T in case of warnings or errors, T in case of warnings or errors excluding style-warnings.

(compile-file-pathname file [output-file path] [other-arguments])

▷ Pathname compile-file writes to if invoked with the same arguments.

(load path

(verbatim (load-varname)

(print (verbatim (load-print)

(if (verbatim (load-verbatim)

(verbatim (load-verbatim)

(setf name definition))))

▷ Load source file or compiled file into Lisp environment. Return T if successful.

(part-name string)

▷ Returns name, package, property list, value, or function, respectively, of symbol.

(setf documentation new-doc)

▷ Get/set documentation string of foo of given type.

nil()

▷ Truth; the supertype of every type including T; the superclass of every class except T, *terminal-io*.

14.4 Standard Packages

common-lisp

▷ Exports the defined names of Common Lisp except for those in the keyword package.

common-lisp-user|cl-user

▷ Current package after startup; uses package common-lisp.

keyword

▷ Contains symbols which are defined to be of type keyword.

15 Compiler

15.1 Predicates

(special-operator-p foo) ▷ T if foo is a special operator.

(compiled-function-p foo) ▷ T if foo is of type compiled-function.

15.2 Compilation

(compile (NIL definition

(setf name definition))

▷ Return compiled function or replace name’s function definition with the compiled function. Return T in case of warnings or errors, and NIL in case of warnings or errors excluding style-warnings.

(compile-file file

(verbatim (compile-varname)

(print file (compile-print)

(output-file out-path

(verbatim (compile-verbatim)

(print (verbatim (compile-print)

(verbatim (compile-verbatim)

(setf name definition)))

▷ Write compiled contents of file to out-path. Return true output path or NIL, T in case of warnings or errors, T in case of warnings or errors excluding style-warnings.

(compile-file-pathname file [output-file path] [other-arguments])

▷ Pathname compile-file writes to if invoked with the same arguments.

(load path

(verbatim (load-varname)

(print (verbatim (load-print)

(if (verbatim (load-verbatim)

(verbatim (load-verbatim)

(setf name definition))))

▷ Load source file or compiled file into Lisp environment. Return T if successful.

(part-name string)

▷ Returns name, package, property list, value, or function, respectively, of symbol.

(setf documentation new-doc)

▷ Get/set documentation string of foo of given type.

nil()

▷ Truth; the supertype of every type including T; the superclass of every class except T, *terminal-io*. 

14.4 Standard Packages

common-lisp

▷ Exports the defined names of Common Lisp except for those in the keyword package.

common-lisp-user|cl-user

▷ Current package after startup; uses package common-lisp.

keyword

▷ Contains symbols which are defined to be of type keyword.
14 Packages and Symbols

The Loop Facility provides additional means of symbol handling; see loop, page 21.

14.1 Predicates

(symbolp foo)  \(\Rightarrow\)  T if foo is of indicated type.

(packagep foo) \(\Rightarrow\) T if foo is of registered package.

(keywordp foo) \(\Rightarrow\) T if foo is of package.

14.2 Packages

obar|keyword:obar \(\Rightarrow\) Keyword, evaluates to :obar.

package:symbol \(\Rightarrow\) Exported symbol of package.

package::symbol \(\Rightarrow\) Possibly unexported symbol of package.

(defpackage foo
  (:nicknames nick*)
  (:documentation string)
  (:intern intern-symbol*)
  (:use used-package*)
  (:import-from pkg imported-symbol*)
  (:shadowing-import-from pkg shd-symbol*)
  (:shadow shd-symbol*)
  (:export exported-symbol*)
  (:size int)) \(\Rightarrow\) Create or modify package foo with interned-symbols, symbols from used-packages, imported-symbols, and shd-symbols. Add shd-symbols to foo’s shadowing list.

(make-package foo
  (:nicknames (nick*) \#:\)) \(\Rightarrow\) Create package foo.

(rename-package package new-name \#:\ new-nicknames \#:\) \(\Rightarrow\) Rename package. Return renamed package.

(in-package foo) \(\Rightarrow\) Make package foo current.

(use-package \#:\ other-packages \#:\ package \#:\) \(\Rightarrow\) Make exported symbols of other-packages available in package, or remove them from package, respectively. Return T.

(package-use-list package)

(package-used-by-list package) \(\Rightarrow\) List of other packages used by/using package.

(delete-package package)

*package* \(\Rightarrow\) The current package.

(list-all-packages) \(\Rightarrow\) List of registered packages.

(package-name package) \(\Rightarrow\) Name of package.

(package-nicknames package) \(\Rightarrow\) Nicknames of package.

(find-package name) \(\Rightarrow\) Package with name (case-sensitive).

(find-all-symbols foo) \(\Rightarrow\) List of symbols foo from all registered packages.

14.3 Symbols

A symbol has the attributes name, home package, property list, and optionally value (of global constant or variable name) and function (function, macro, or special operator name).

(make-symbol name) \(\Rightarrow\) Make fresh, uninterned symbol name.

(gensym \#:) \(\Rightarrow\) Return fresh, uninterned symbol #\$n with n from *gensym-counter*. Increment *gensym-counter*.

(gentemp \#:\ prefix \#:\ package \#:\) \(\Rightarrow\) Intern fresh symbol in package. Deprecated.

(copy-symbol symbol \#:\ props) \(\Rightarrow\) Return uninterned copy of symbol. If props is T, give copy the same value, function and property list.
\( \text{macroexpand-1} \) \( \text{form} \) (environment \( \text{NIL} \))

- Return macro expansion, once or entirely, respectively, of \( \text{form} \) and \( \text{NIL} \) if \( \text{form} \) was a macro form. Return \( \text{form} \) and \( \text{NIL} \) otherwise.

\( \text{macroexpand} \) \( \text{form} \)

- Function of arguments expansion function, macro form, and environment called by \( \text{macroexpand-1} \) to generate macro expansions.

\( \text{trace} \) \( \text{function} \) (\( \text{setf} \) \( \text{function} \))

- Cause functions to be traced. With no arguments, return list of traced functions.

\( \text{untrace} \) \( \text{function} \) (\( \text{setf} \) \( \text{function} \))

- Stop functions, or each currently traced function, from being traced.

\( \text{trace-output} \)

- Output stream \( \text{trace} \) and \( \text{time} \) send their output to.

\( \text{step} \) \( \text{form} \)

- Step through evaluation of \( \text{form} \). Return values of \( \text{form} \).

\( \text{break} \) \( \text{control-arg} \)

- Jump directly into debugger; return \( \text{NIL} \). See page 36, \( \text{format} \), for control and args.

\( \text{time} \)

- Evaluate \( \text{forms} \) and print timing information to \( \text{trace-output} \). Return values of \( \text{form} \).

\( \text{inspect} \) \( \text{foo} \)

- Interactively give information about \( \text{foo} \).

\( \text{describe} \) \( \text{foo} \) \( \text{stream} \) \( \text{standard-output} \)

- Send information about \( \text{foo} \) to \( \text{stream} \). Called by \( \text{describe} \).

\( \text{describe-object} \) \( \text{foo} \) \( \text{stream} \)

- Send information about \( \text{foo} \) to \( \text{stream} \). Called by \( \text{describe} \).

\( \text{disassemble} \) \( \text{function} \)

- Send disassembled representation of \( \text{function} \) to \( \text{standard-output} \). Return \( \text{NIL} \).

\( \text{room} \) \( \text{XIL} \) \( \text{default} \) \( \text{NIL} \)

- Print information about internal storage management to \( \text{standard-output} \).

15.4 Declarations

\( \text{proclaim} \) \( \text{decl} \)

- Globally make declaration(s) \( \text{decl} \). \( \text{decl} \) can be: declaration, type, ftype, inline, notinline, optimize, or special. See below.

\( \text{declare} \) \( \text{decl} \)

- Inside certain forms, locally make declarations \( \text{decl} \). \( \text{decl} \) can be: dynamic-extent, type, ftype, ignorable, ignore, inline, notinline, optimize, or special. See below.

\( \text{declaration} \) \( \text{foo} \)

- Make \( \text{foo} \) names of declarations.

\( \text{dynamic-extent} \) \( \text{variable} \) \( \text{function} \) \( \text{function} \)

- Declare lifetime of variables and/or functions to end when control leaves enclosing block.

\( \text{type} \) \( \text{variable} \)

- Declare variables or functions to be of type.

\( \text{ignore} \) \( \text{function} \)

- Suppress warnings about used/unused bindings.

(inline function*)

(notinline function*)

- Tell compiler to integrate/not to integrate, respectively, called functions into the calling routine.

(compile-speed)

- Tell compiler how to optimize. \( n = 0 \) means unimportant, \( n = 1 \) is neutral, \( n = 3 \) means important.

(special \( \text{var} \))

- Declare \( \text{vars} \) to be dynamic.

16 External Environment

\( \text{get-internal-real-time} \)

- Current time, or computing time, respectively, in clock ticks.

\( \text{internal-time-units-per-second} \)

- Number of clock ticks per second.

\( \text{encode-universal-time} \) \( \text{sec} \) \( \text{min} \) \( \text{hour} \) \( \text{date} \) \( \text{month} \) \( \text{year} \) \( \text{zone} \)

- Seconds from 1900-01-01, 00:00, ignoring leap seconds.

\( \text{decode-universal-time} \) \( \text{universal-time} \) \( \text{time-zone} \)

- Return second, minute, hour, date, month, year, day, daylight-p, and zone.

\( \text{short-site-name} \)

\( \text{long-site-name} \)

- String representing physical location of computer.

\( \text{lisp-implementation} \)

\( \text{software} \)

\( \text{machine} \) \( \text{version} \)

- Name or version of implementation, operating system, or hardware, respectively.

\( \text{machine-instance} \)

- Computer name.